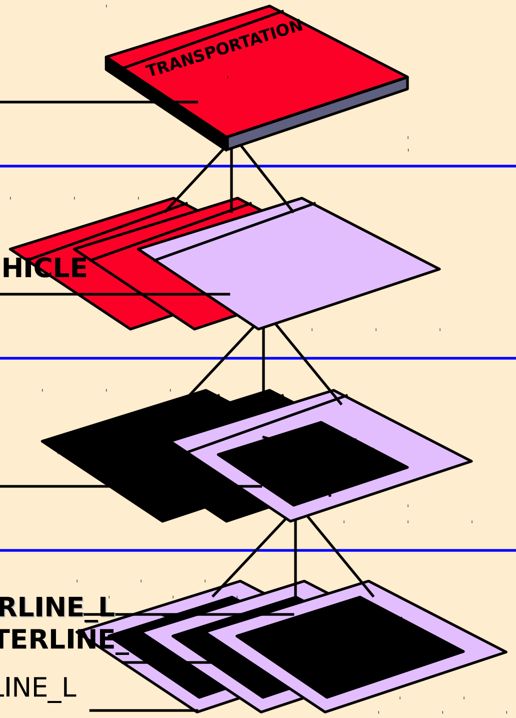
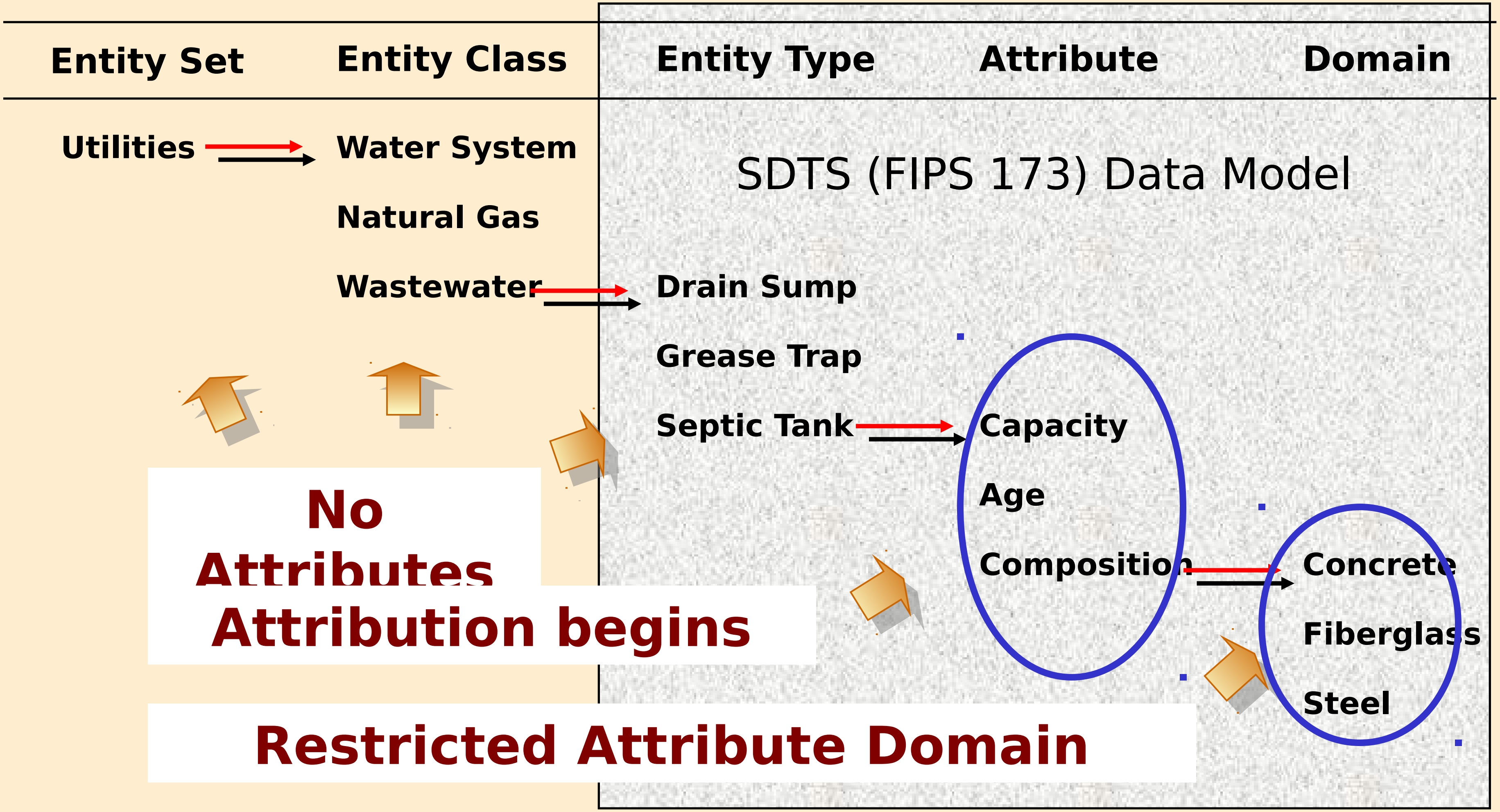
UML
Object Modeling

Data Model Terminology



DATA HIERARCHY	EXAMPLE DATA	GIS		CADD AM/FM	
		MGE	ARC/INFO	MicroStation	AutoCAD
ENTITY SET	TRANSPORTATION	PROJECT LEVEL	PROJECT LEVEL	PROJECT LEVEL	PROJECT LEVEL
ENTITY CLASS	TRANSPORTATION_VEHICLE	CATEGORY AND DESIGN FILE	WORKSPACE	DESIGN FILE	DRAWING FILE
ENTITY TYPE	ROAD CENTERLINE	GROUP BY FEATURES	COVERAGE FILE	GROUP BY LEVEL	GROUP BY LAYER
ENTITY	PRIMARY_ROAD_CENTERLINE_L SECONDARY_ROAD_CENTERLINE_ TERTIARY_ROAD_CENTERLINE_L	FEATURE	SELECT BY ATTRIBUTE	LEVEL	LAYER

Data Model Example



Entity Set	Entity Class	Entity Type	Attribute	Domain
Utilities	Water System			
	Natural Gas			
	Wastewater	Drain Sump		
		Grease Trap		
		Septic Tank	Capacity	
			Age	
			Composition	Concrete
				Fiberglass
				Steel

SDTS (FIPS 173) Data Model

No Attributes

Attribution begins

Restricted Attribute Domain

Model Organization

not just data
but the
process

Object Model

Visual Modeling:

Process of graphically depicting the **system** to be developed.

Allows user to present the essential details of a complex problem and filter out non-essential details.

Provides a mechanism for viewing the entire system from different perspectives.

A View is a look at a certain part of a Model within a specified criteria.

The View can use various diagrams to represent the View.

The diagram is a specific application of the View:

Views:

Use Case:

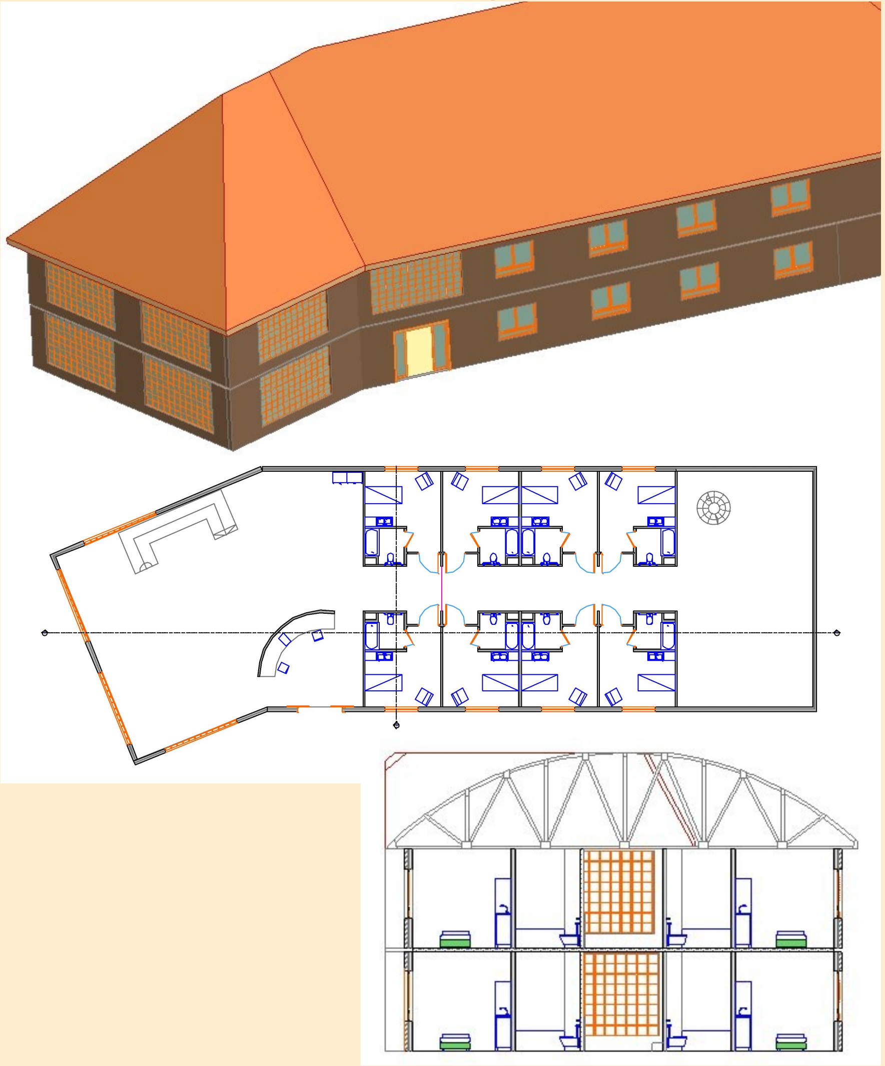
Logical:

Component:

Deployment:

Model Views

Visual Modeling:



Views of a Building model are:

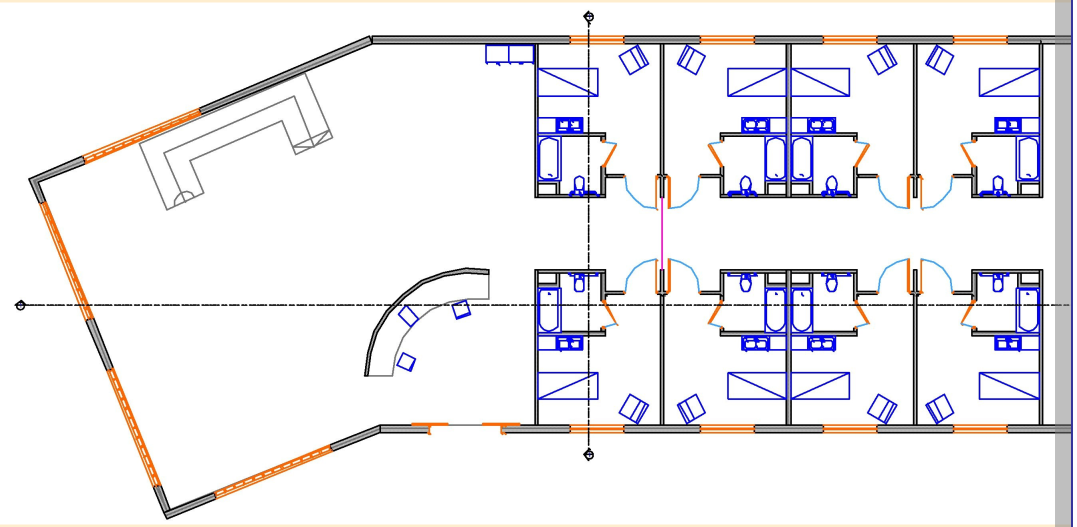
- The floor plan
- Section
- Elevation
- Details
- Roof plan
- …

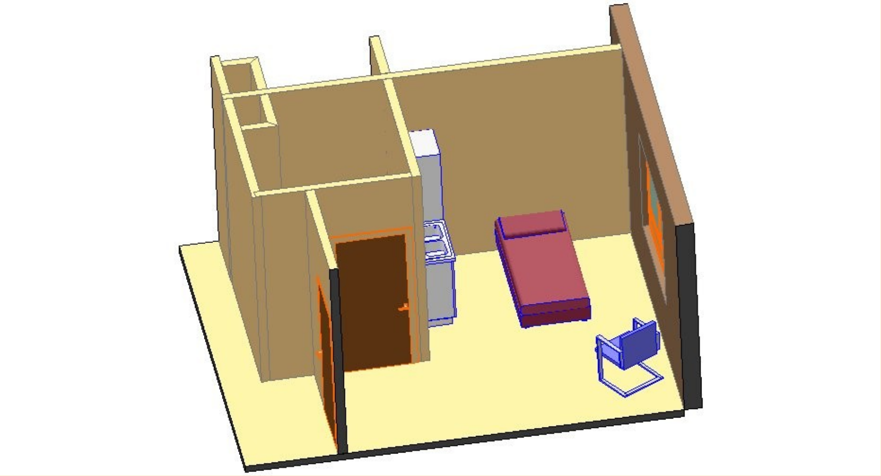
Each view allows the User to look at the model with a certain focus (the layout and types of walls on the fourth floor, the reflected ceiling plan).

A model must have views to be designed and edited.

Model Views

Visual Modeling:





Views of a Building model allow the User to focus on areas such as the floor plan.

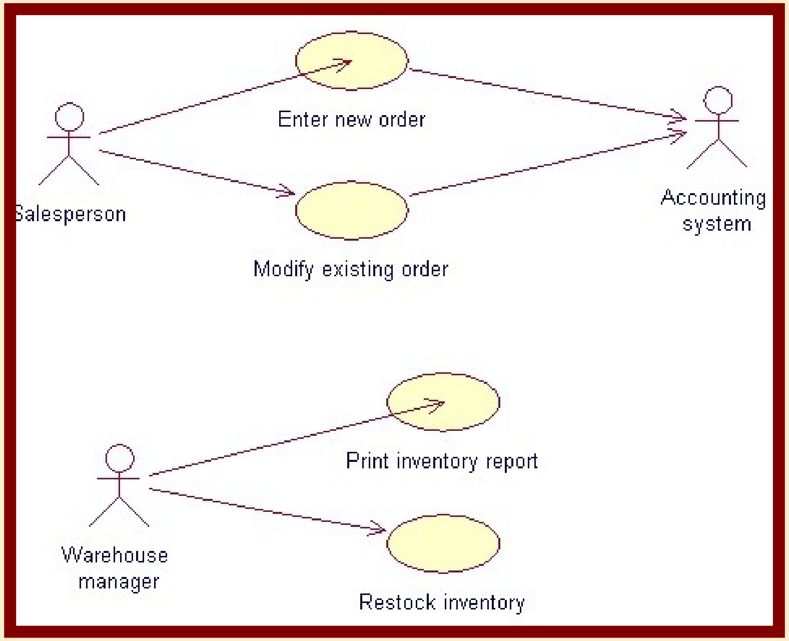
The floor plan shows the information essential to constructing the walls and fixed equipment.

A User defined 3D Model view of a portion show how the objects fit together.

The Floor plan is considered a Use Case of the model

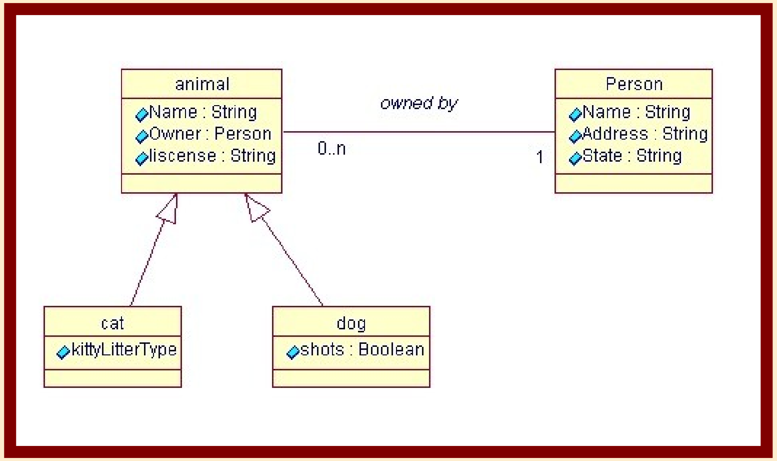
A consolidated model must be editable in all views.

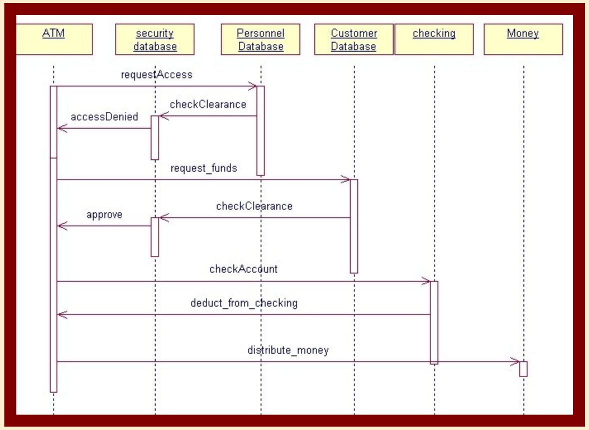
UML Model Design Approach



Use Case Diagram

Class Diagram





Sequence Diagram

View Diagrams

Visual Modeling:

Use Case: see how actors and use cases interact

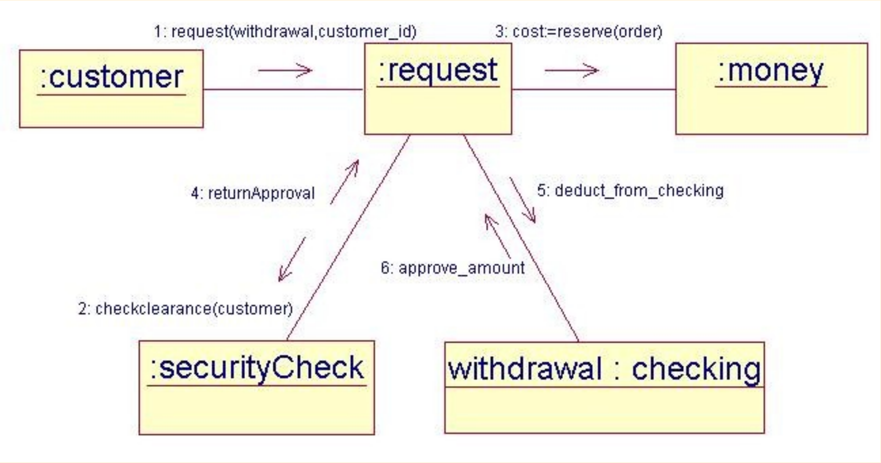
Diagrams:

Use-Case diagrams

Sequence diagrams

Collaboration diagrams

Activity diagrams

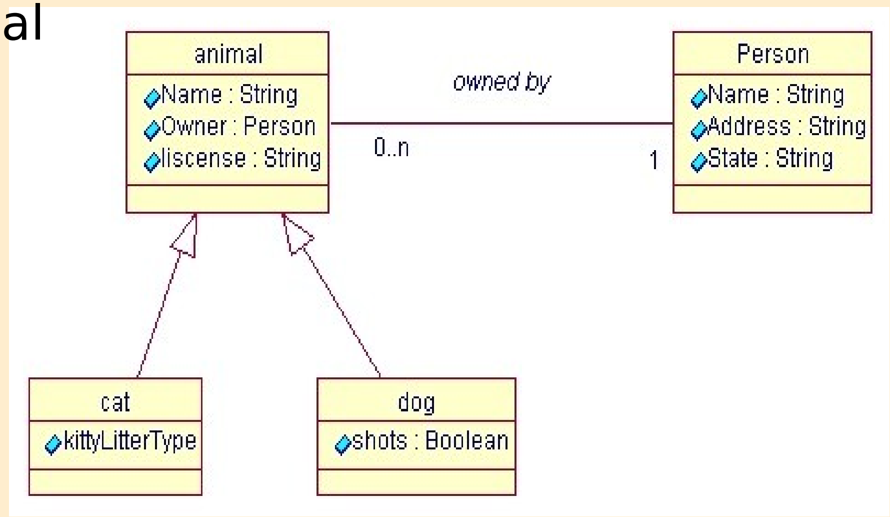


Logical View: addresses functional requirements: what it takes to accomplish the Use Case. Looks at Classes and their relationships.

Diagrams:

Class diagram

Statechart diagram



View Diagrams

Visual Modeling:

Component View: addresses software organization of the system. Information about the software, the executable, and the library components for the system.

Diagrams:

Main (by default)

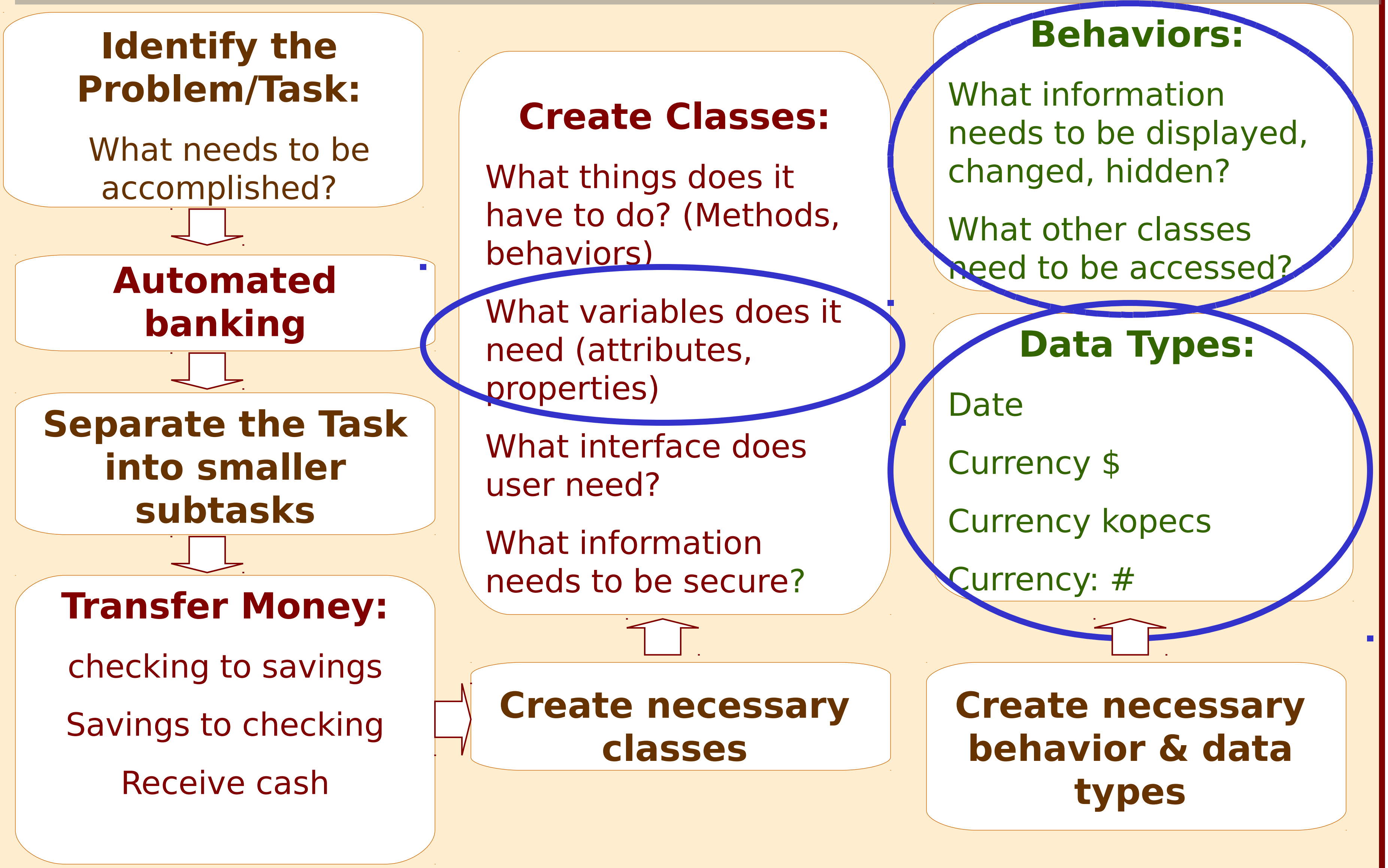
Additional diagrams can be added to this view throughout the analysis and design process.

Deployment View: Shows the mapping of the processes to the actual hardware. For distributed architecture environment with Servers and Applications at different locations.

Diagram

Deployment Diagram

Model Design Process



Identify the Problem/Task:

What needs to be accomplished?

Automated banking

Separate the Task into smaller subtasks

Transfer Money:

checking to savings

Savings to checking

Receive cash

Create necessary classes

Create Classes:

What things does it have to do? (Methods, behaviors)

What variables does it need (attributes, properties)

What interface does user need?

What information needs to be secure?

Create necessary behavior & data types

Behaviors:

What information needs to be displayed, changed, hidden?

What other classes need to be accessed?

Data Types:

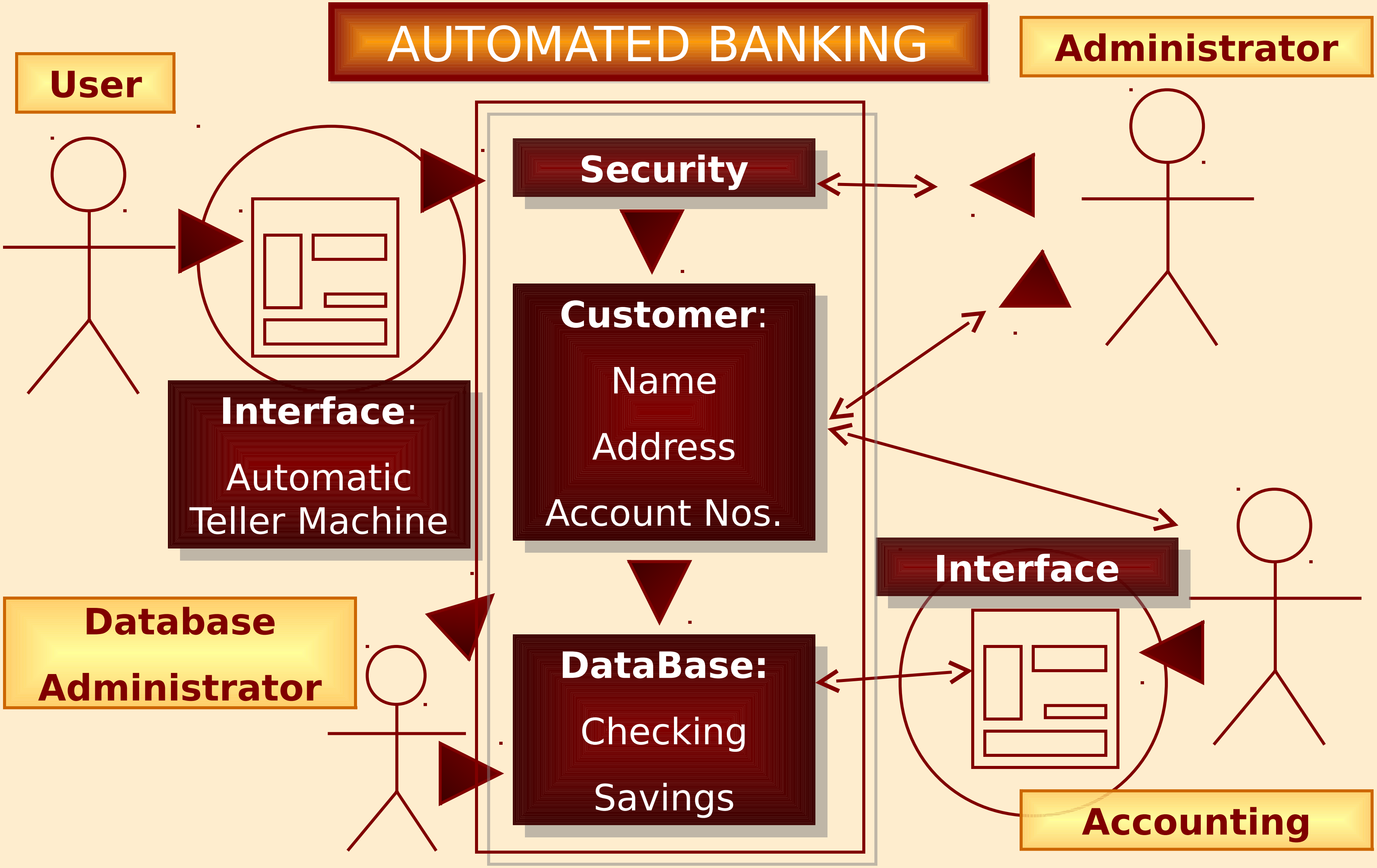
Date

Currency $

Currency kopecs

Currency: #

Model : Use Case



AUTOMATED BANKING

User

Administrator

Security

Customer:

Name

Address

Account Nos.

Interface:

Automatic Teller Machine

Interface

Database Administrator

DataBase:

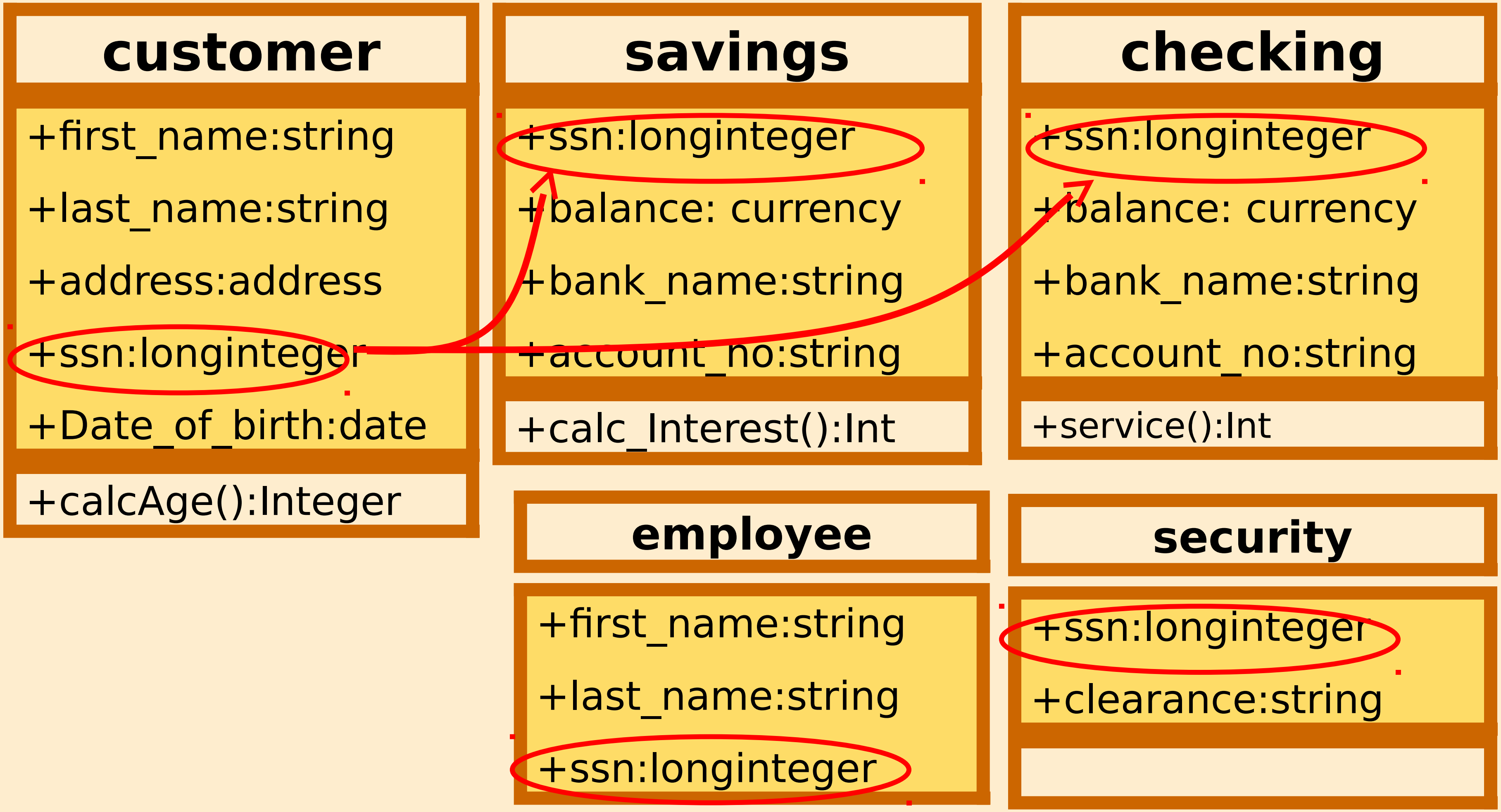
Checking

Savings

Accounting

Model Class Information

Banking information classes



customer

+first_name:string

+last_name:string

+address:address

+ssn:longinteger

+Date_of_birth:date

+calcAge():Integer

savings

+ssn:longinteger

+balance: currency

+bank_name:string

+account_no:string

+calc_Interest():Int

checking

+ssn:longinteger

+balance: currency

+bank_name:string

+account_no:string

+service():Int

employee

+first_name:string

+last_name:string

+ssn:longinteger

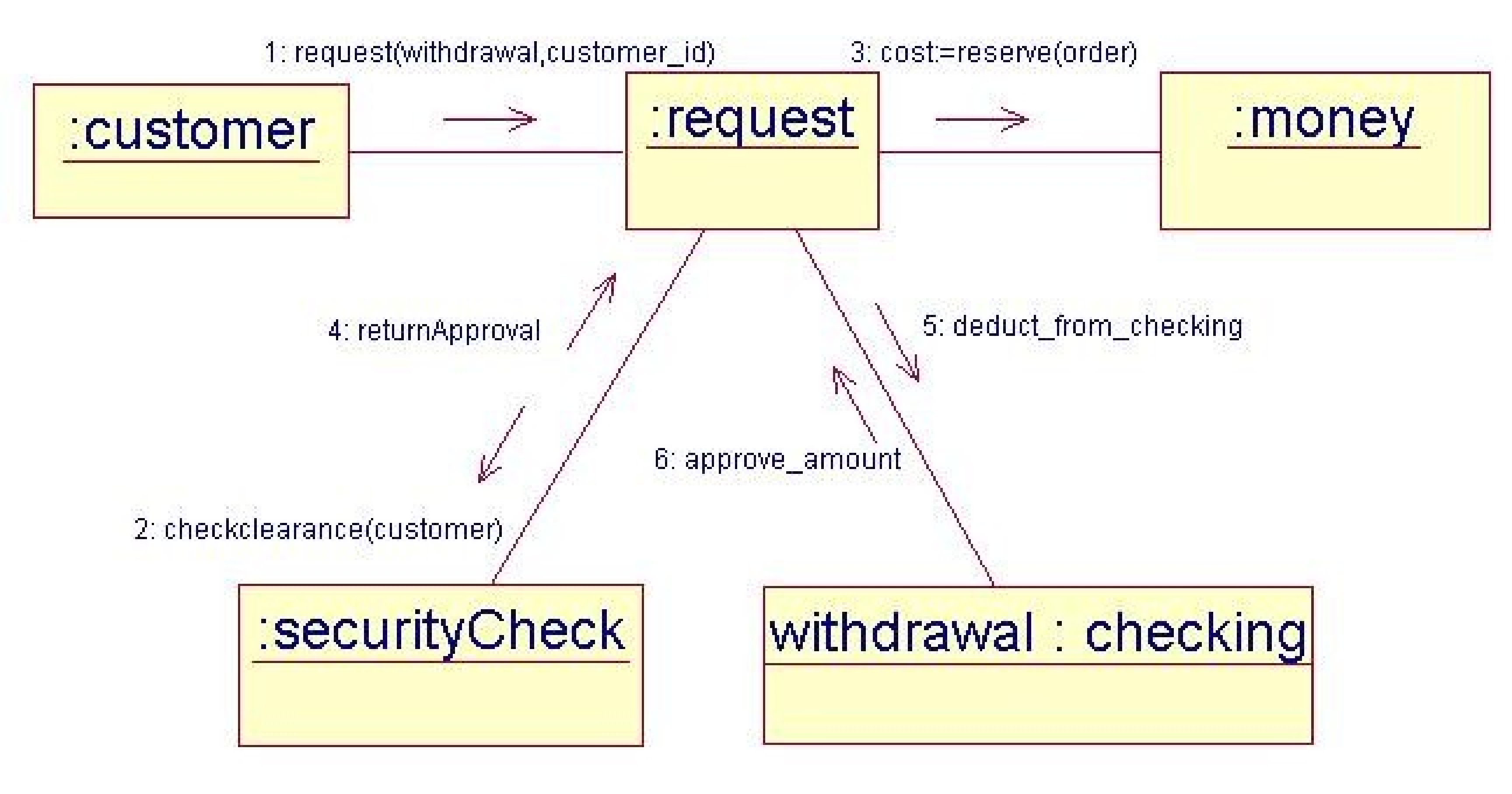
security

+ssn:longinteger

+clearance:string

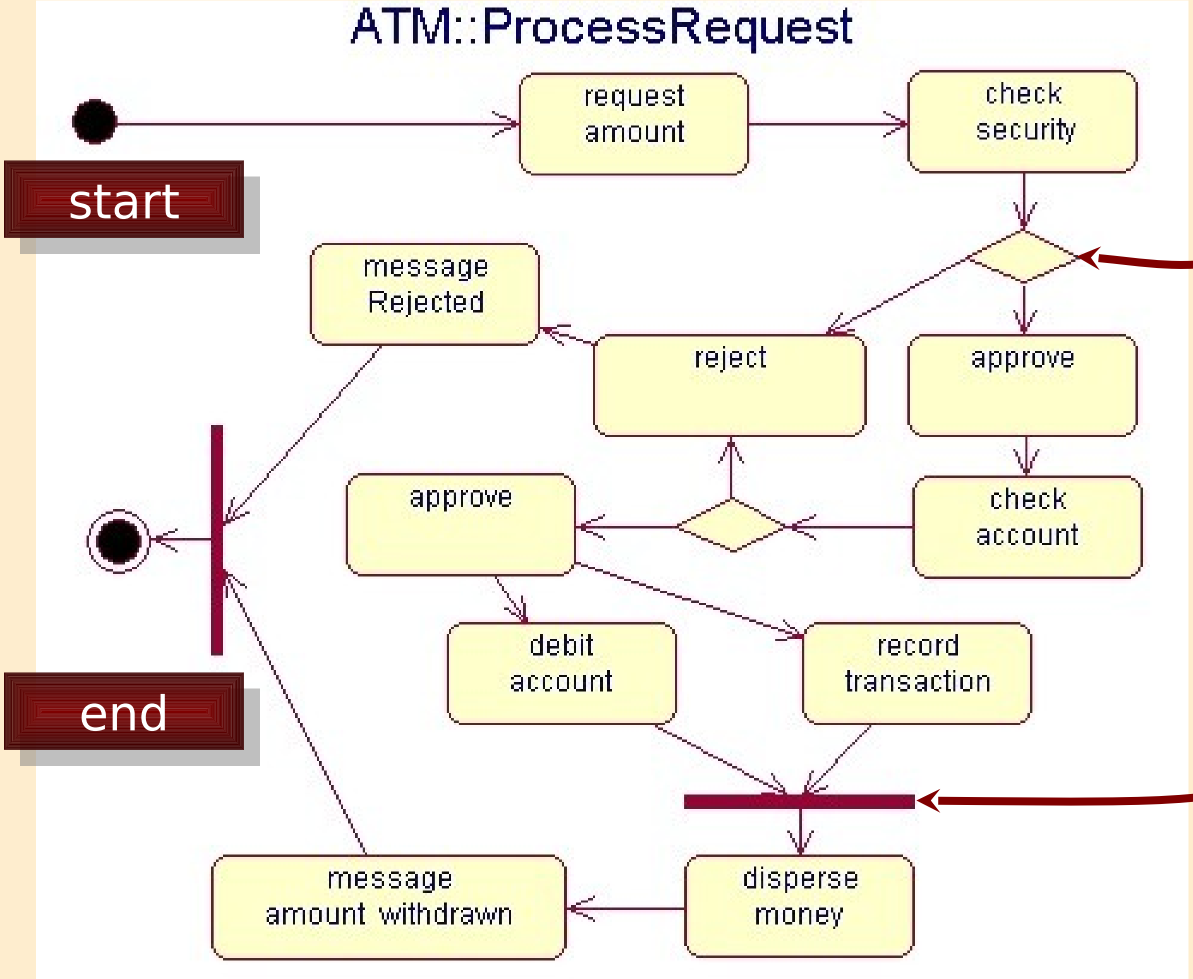
Model Design Process

Collaboration



Model Design Process

State/Activity Diagram



start

end

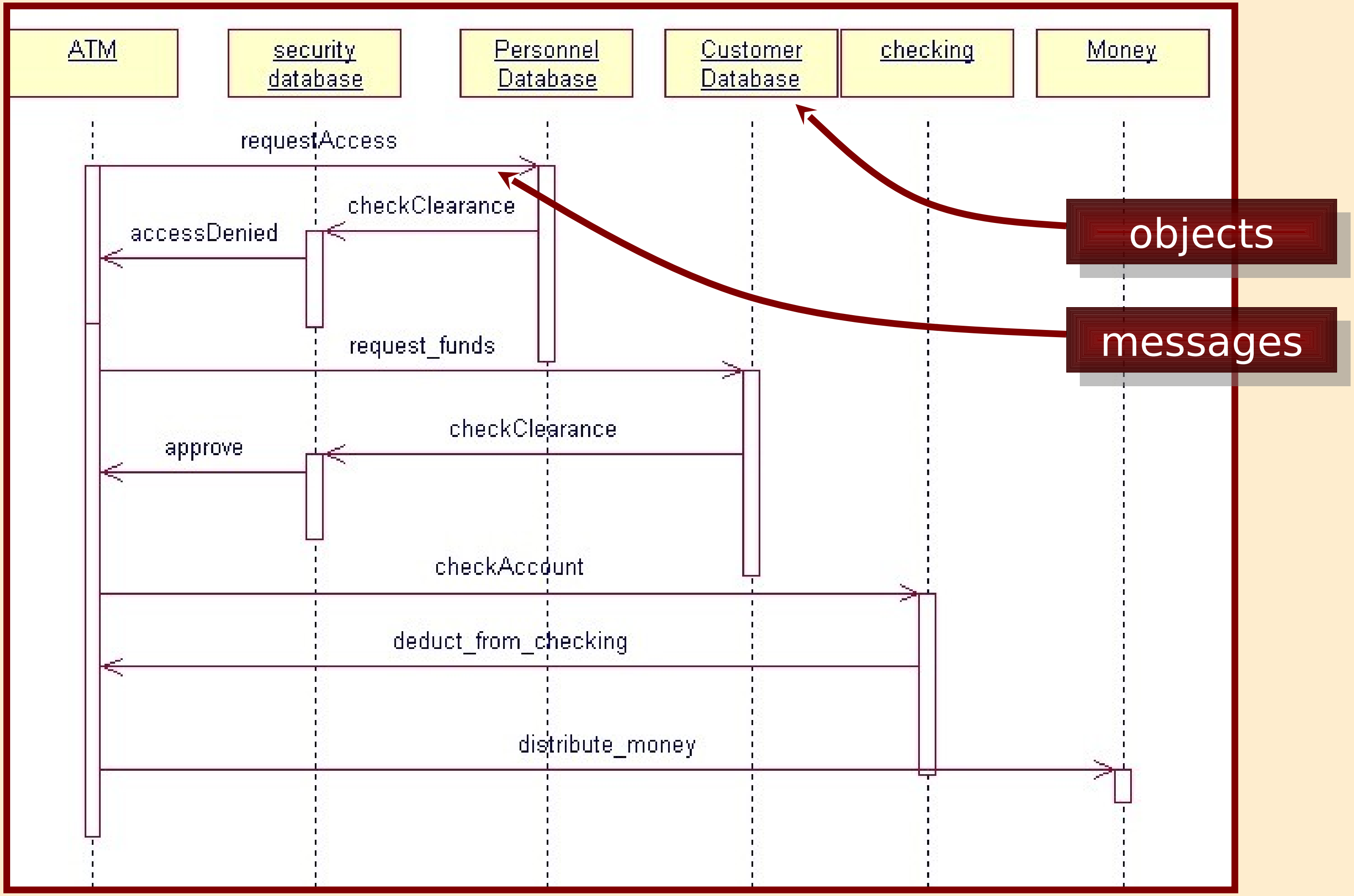
decision

synchronization

Shows the sequence of messages sent in real time.

Sequence Diagram

Sequence Diagram

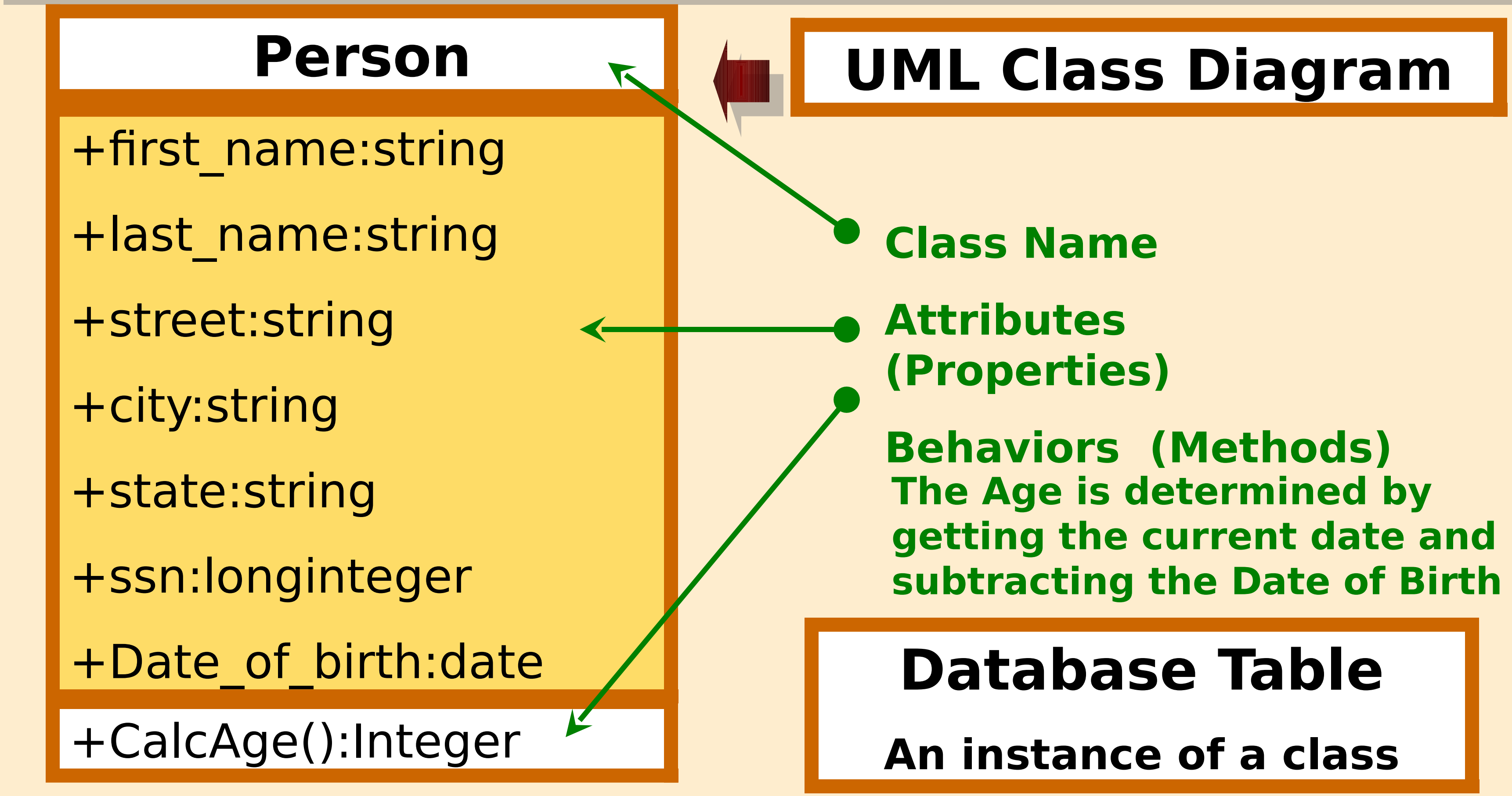


objects

messages

Explanation of Model Classes

UML : Class Structure



Person

+first_name:string

+last_name:string

+street:string

+city:string

+state:string

+ssn:longinteger

+Date_of_birth:date

+CalcAge():Integer

UML Class Diagram

Class Name

Attributes (Properties)

Behaviors (Methods)

The Age is determined by getting the current date and subtracting the Date of Birth

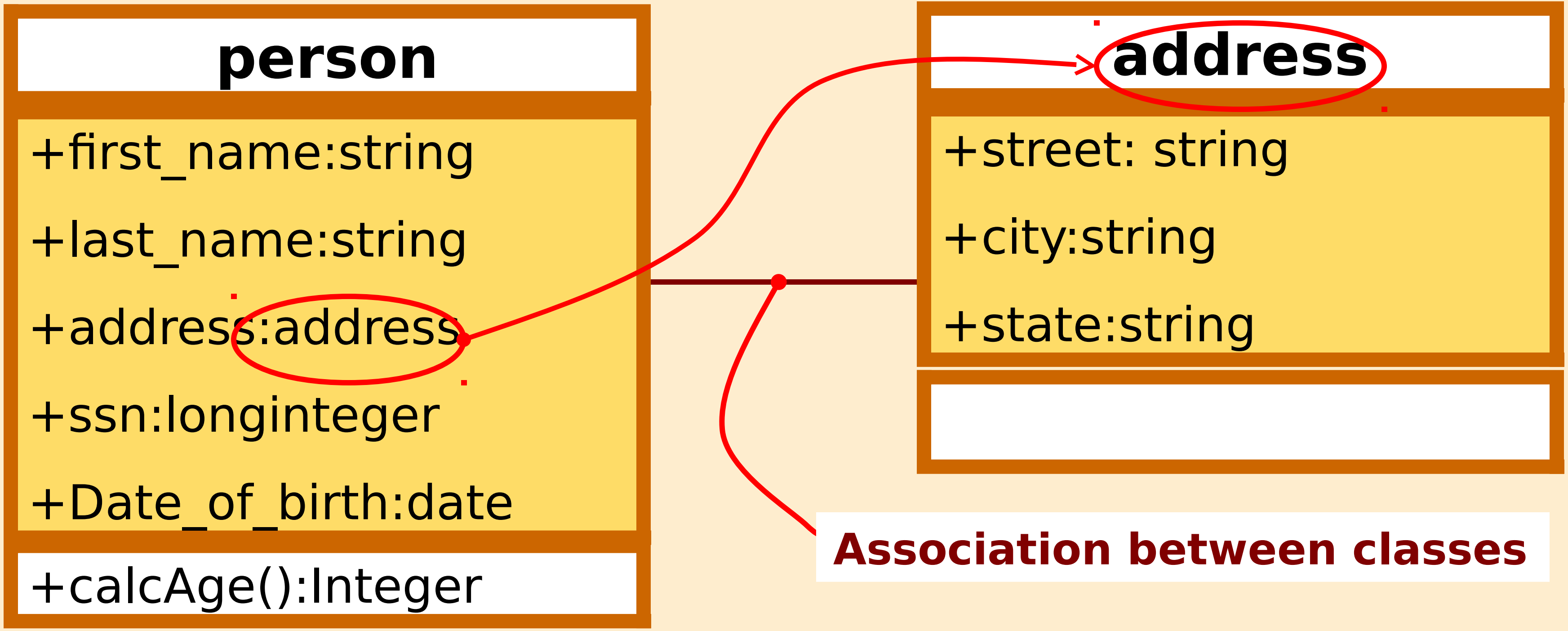
Database Table

An instance of a class

Person

first_name	last_name	street	city	state	ssn	Date_of_birth
Joe	Jones	South	Vicksburg	Ms	1010-010-010	12/12/1986
Sam	Sims	East	Bovina	Ms	1212-121-121	5/2/1964

UML CLASS Inheritance



person

+first_name:string

+last_name:string

+address:address

+ssn:longinteger

+Date_of_birth:date

+calcAge():Integer

address

+street: string

+city:string

+state:string

Association between classes

Produces this information

Database Table

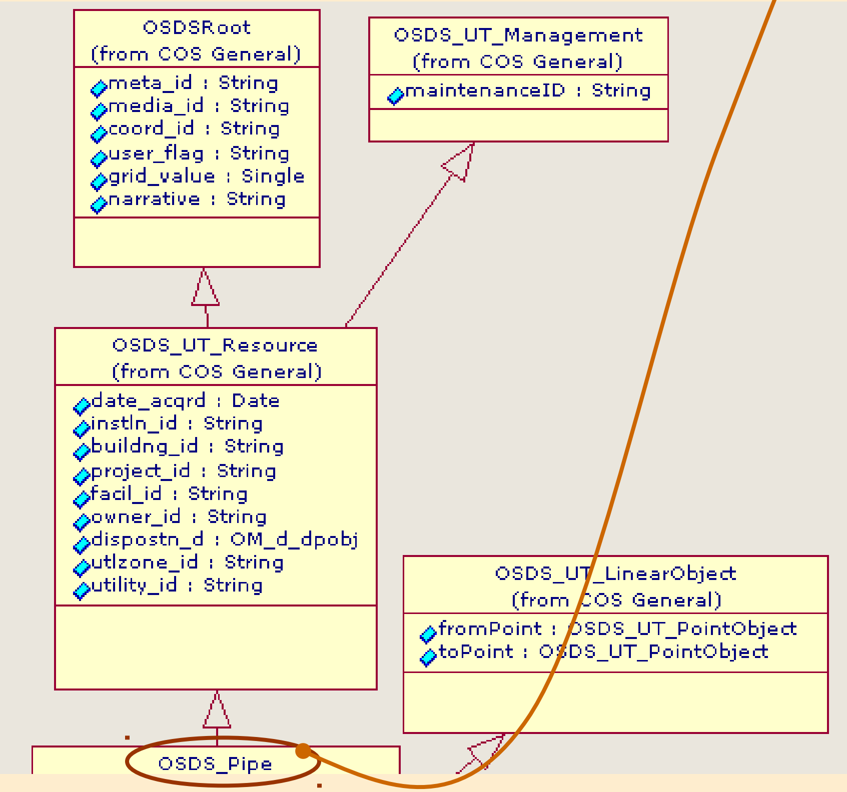
Person

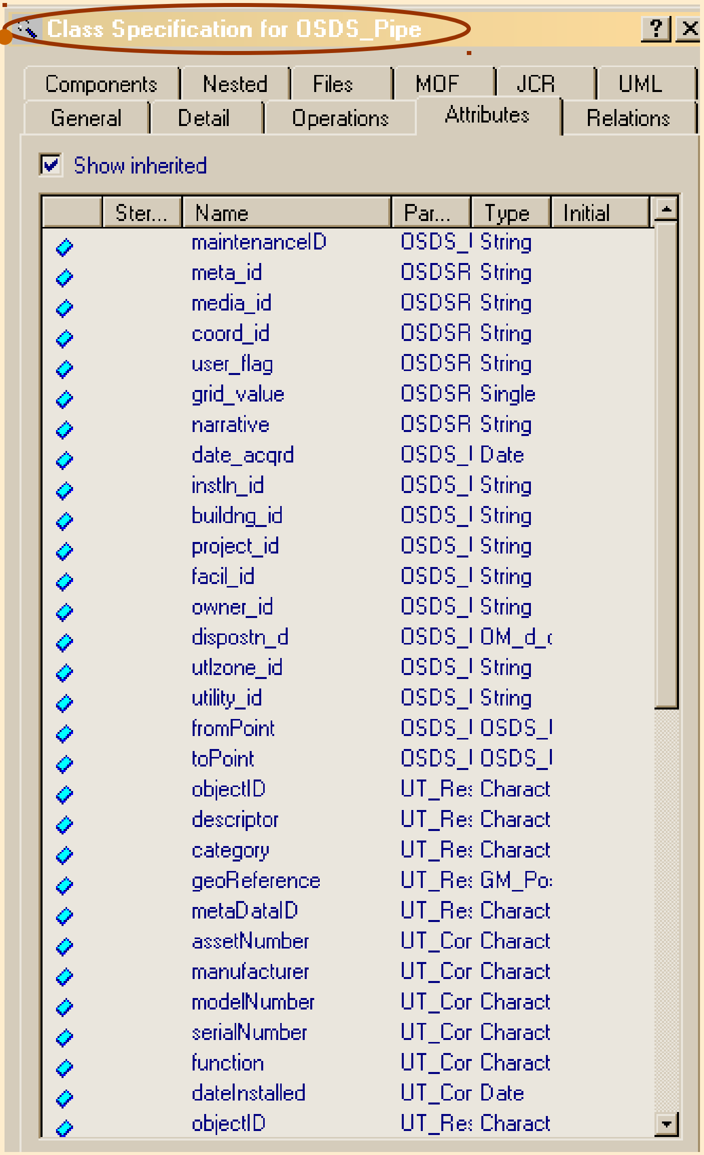
first_name	last_name	street	city	state	ssn	Date_of_birth
Joe	Jones	South	Vicksburg	Ms	1010-010-010	12/12/1986
Sam	Sims	East	Bovina	Ms	1212-121-121	5/2/1964

Class Inheritance

Will create a database table that contains all the information that the above classes have.

Object SDS Pipe contains all the attribute information from OSDS_UT_Resource, Management and Root

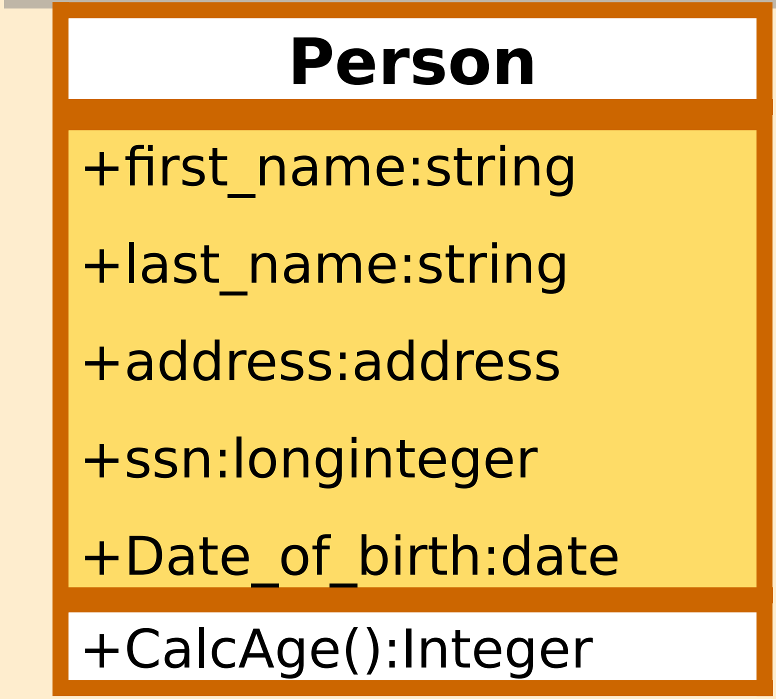




☑ Show inherited

	Ster...	Name	Par...	Type	Initial
◆		maintenanceID	OSDS_I	String	
◆		meta_id	OSDSR	String	
◆		media_id	OSDSR	String	
◆		coord_id	OSDSR	String	
◆		user_flag	OSDSR	String	
◆		grid_value	OSDSR	Single	
◆		narrative	OSDSR	String	
◆		date_acqrd	OSDS_I	Date	
◆		instln_id	OSDS_I	String	
◆		buildng_id	OSDS_I	String	
◆		project_id	OSDS_I	String	
◆		facil_id	OSDS_I	String	
◆		owner_id	OSDS_I	String	
◆		dispostn_d	OSDS_I	OM_d_(	
◆		utlzone_id	OSDS_I	String	
◆		utility_id	OSDS_I	String	
◆		fromPoint	OSDS_I	OSDS_I	
◆		toPoint	OSDS_I	OSDS_I	
◆		objectID	UT_Re:	Charact	
◆		descriptor	UT_Re:	Charact	
◆		category	UT_Re:	Charact	
◆		geoReference	UT_Re:	GM_Po:	
◆		metaDataID	UT_Re:	Charact	
◆		assetNumber	UT_Cor	Charact	
◆		manufacturer	UT_Cor	Charact	
◆		modelNumber	UT_Cor	Charact	
◆		serialNumber	UT_Cor	Charact	
◆		function	UT_Cor	Charact	
◆		dateInstalled	UT_Cor	Date	
◆		objectID	UT_Re:	Charact	

UML Class, XML Schema, Database



Person

+first_name:string

+last_name:string

+address:address

+ssn:longinteger

+Date_of_birth:date

+CalcAge():Integer

XML Schema

```
<Person>
        <first_name></first_name>
        <last_name> </last_name>
        <street> </street>
        <city> </city>
        <state> </state>
        <ssn> </ssn>
        <date_of_birth>
</date_of_birth>
</Person>
```

Database Table

Person

first_name	last_name	street	city	state	ssn	Date_of_birth
Joe	Jones	South	Vicksburg	Ms	1010-010-010	12/12/1986
Sam	Sims	East	Bovina	Ms	1212-121-121	5/2/1964

Class & Instances of class

class

The specification or schema for the OBJECT

pet

-isSleeping:boolean=true

-isEating:boolean=false

eat()

sleep()

The OBJECT Implementation

with specific data

Instances : Objects

PuddyTat:pet

-isSleeping:boolean=False

-isEating:boolean=true

eat()

sleep()

Fido:pet

-isSleeping:boolean=false

-isEating:boolean=true

eat()

sleep()

UML Nomenclature

Class Relationships

Association

relationship between classes

Generalization

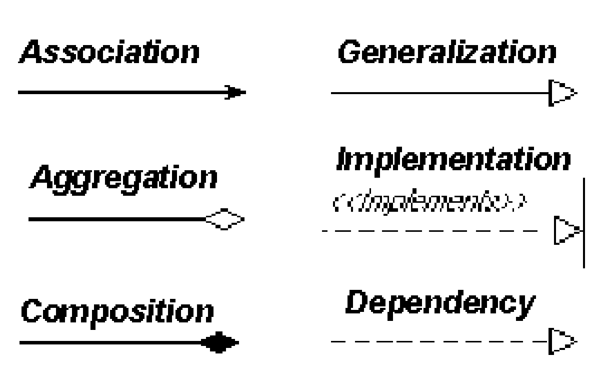
relationship between a more general class and a more specific class: ie SuperClass and Subclass

a “kind-of” relationship.

Composition

is a special type of aggregation that denotes a strong ownership between Class A, the whole, and Class B, its part.

Navigable:



Implementation

the static physical implementation of a class: to point that the Object gets created.

Dependency

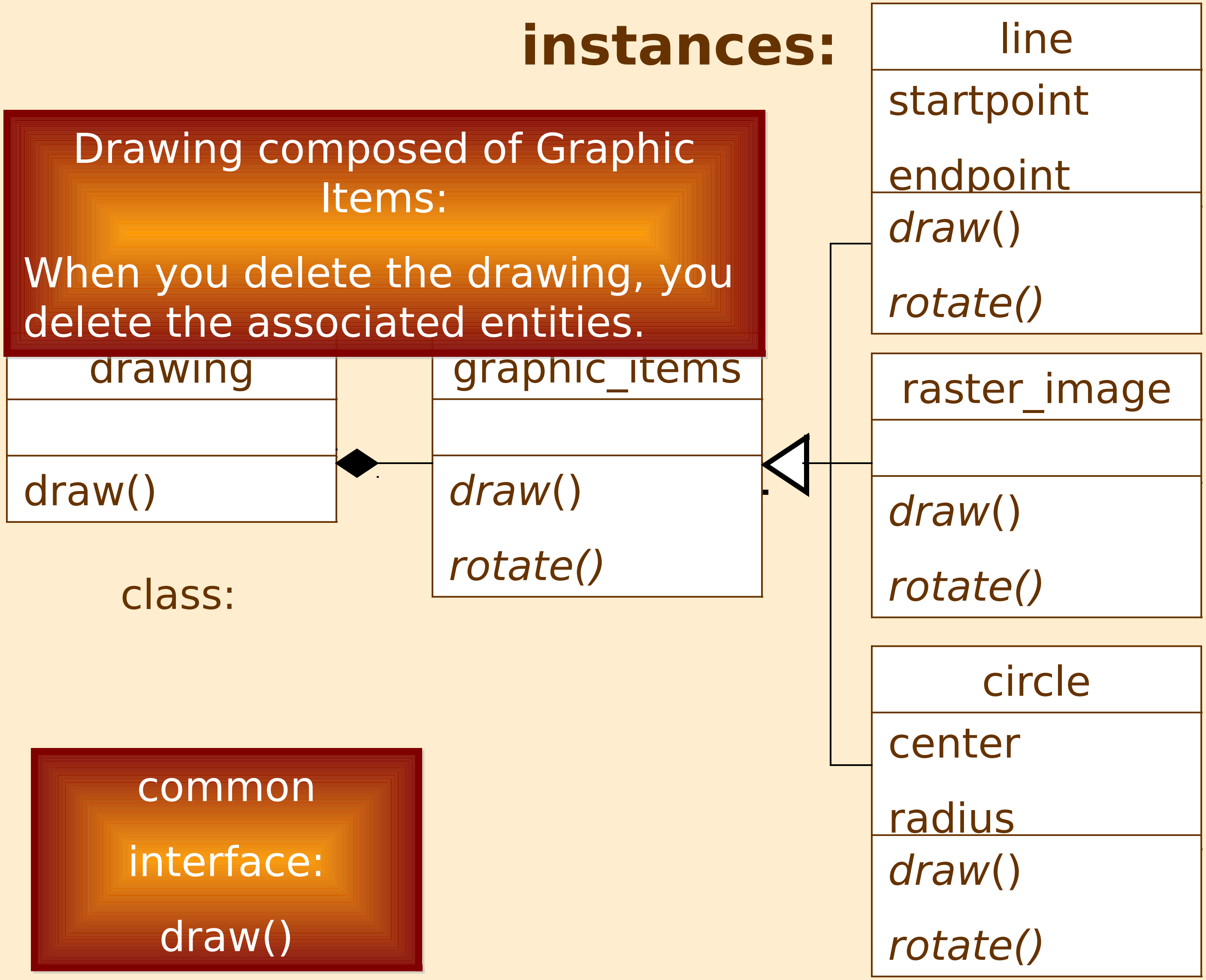
relationship in which a change in the independent element effects the dependent element.

Association

instances:

Drawing composed of Graphic Items:

When you delete the drawing, you delete the associated entities.



drawing

draw()

graphic_items

draw()

rotate()

line

startpoint

endpoint

draw()

rotate()

raster_image

draw()

rotate()

circle

center

radius

draw()

rotate()

class:

common

interface:

draw()

Creates information or changes the information using separate code

Aggregation

Aggregation:

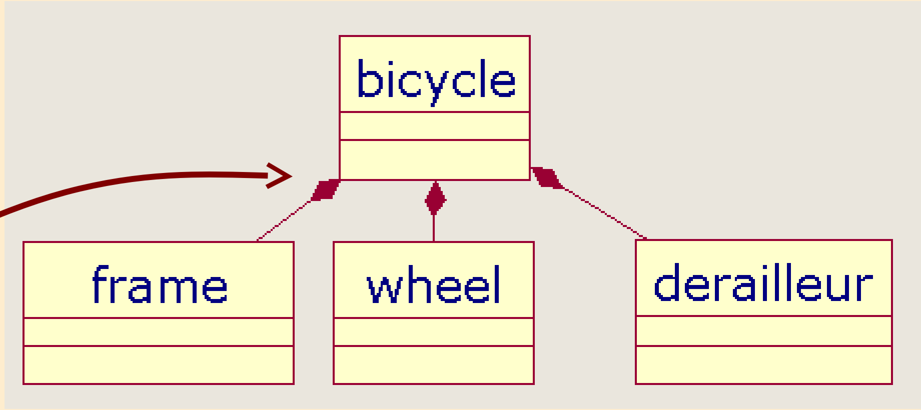
Aggregation association that represents component hierarchy

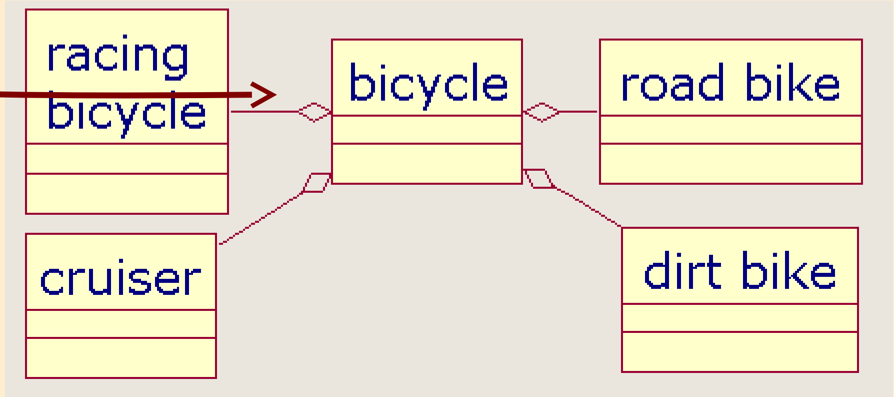
Complex (Composition)

is a special type of aggregation is composed of parts to make a whole.

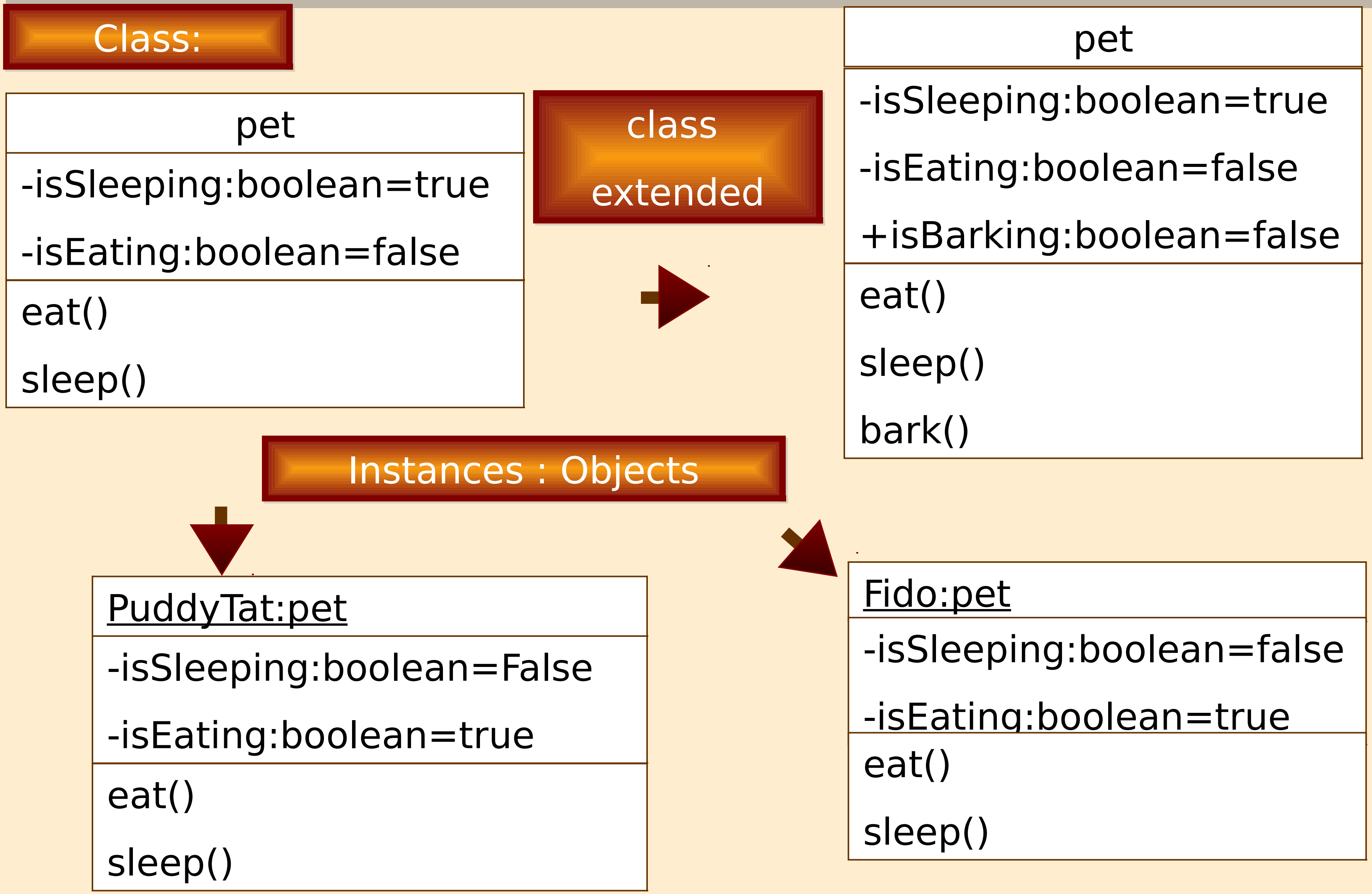
Simple

The main class is a general abstract class, the subclasses form the main types of instances.





Stereotype/subclass



Class:

pet

-isSleeping:boolean=true

-isEating:boolean=false

eat()

sleep()

class extended

pet

-isSleeping:boolean=true

-isEating:boolean=false

+isBarking:boolean=false

eat()

sleep()

bark()

Instances : Objects

PuddyTat:pet

-isSleeping:boolean=False

-isEating:boolean=true

eat()

sleep()

Fido:pet

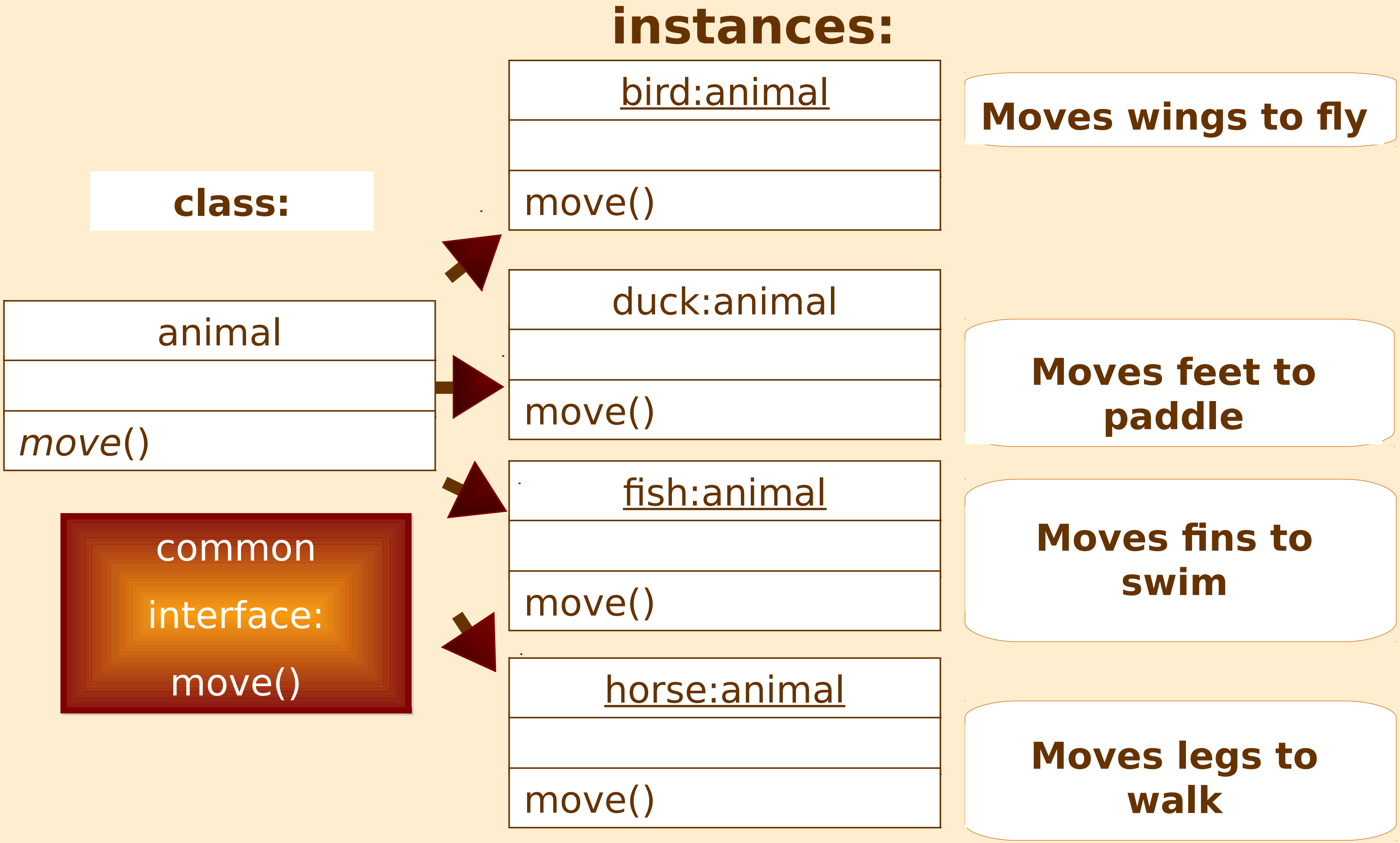
-isSleeping:boolean=false

-isEating:boolean=true

eat()

sleep()

Polymorphic



instances:

class:

animal

move()

common interface: move()

bird:animal

move()

duck:animal

move()

fish:animal

move()

horse:animal

move()

Moves wings to fly

Moves feet to paddle

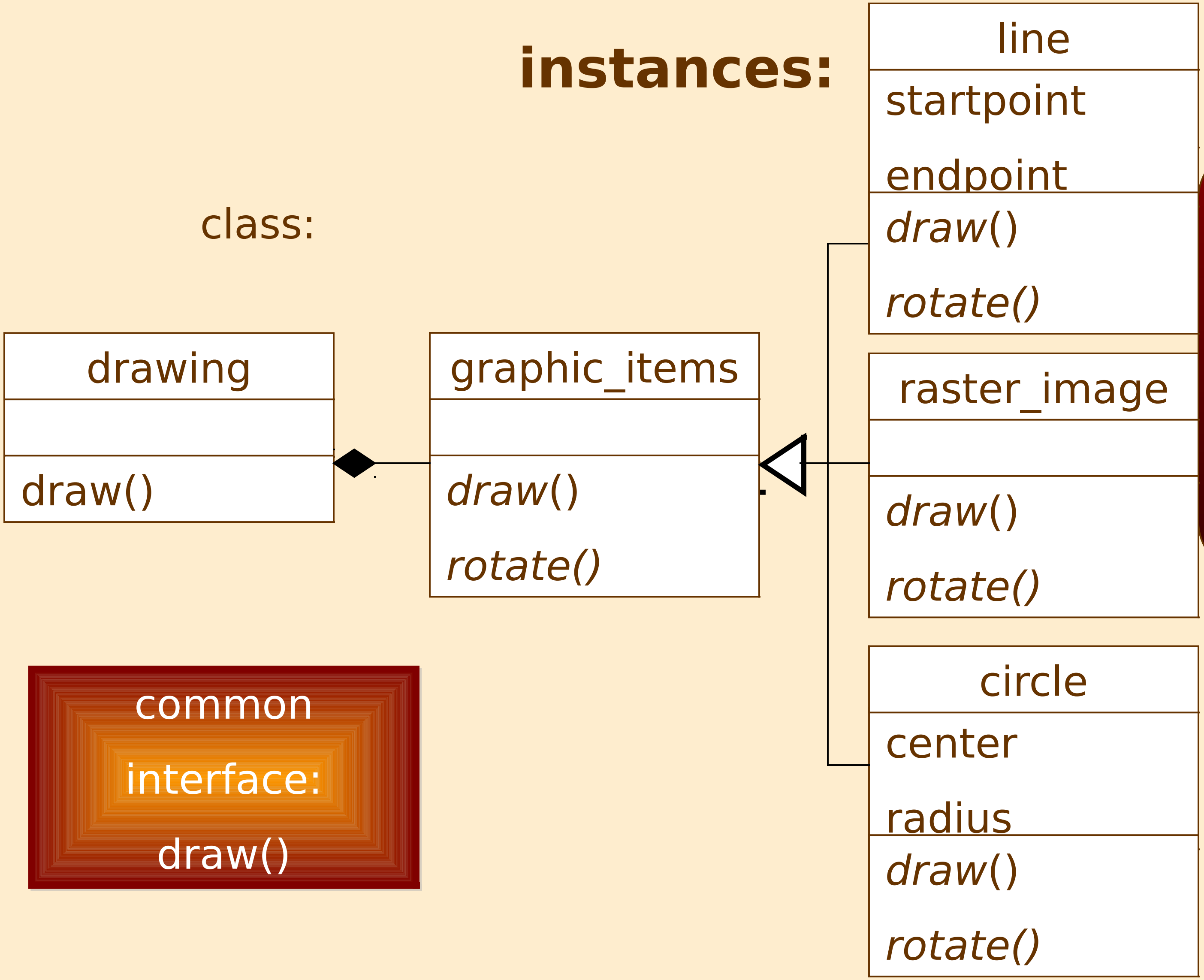
Moves fins to swim

Moves legs to walk

Polymorphic : Graphics

instances:

class:



drawing

draw()

graphic_items

draw()

rotate()

line

startpoint

endpoint

draw()

rotate()

raster_image

draw()

rotate()

circle

center

radius

draw()

rotate()

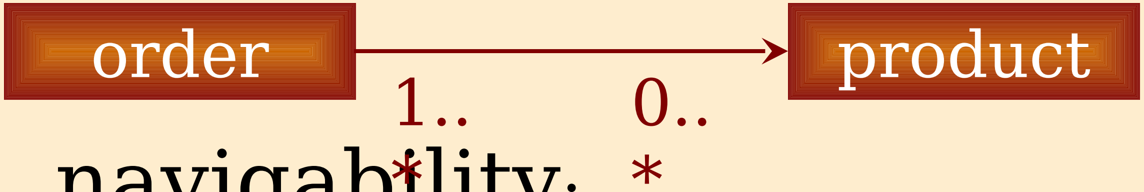
Creates information or changes the information using separate code

common

interface:

draw()

Navigation & Cardinality



navigability:

shows the direction for information flow:

- to make an ORDER, PRODUCTS are selected

cardinality:

shows quantity relationships:

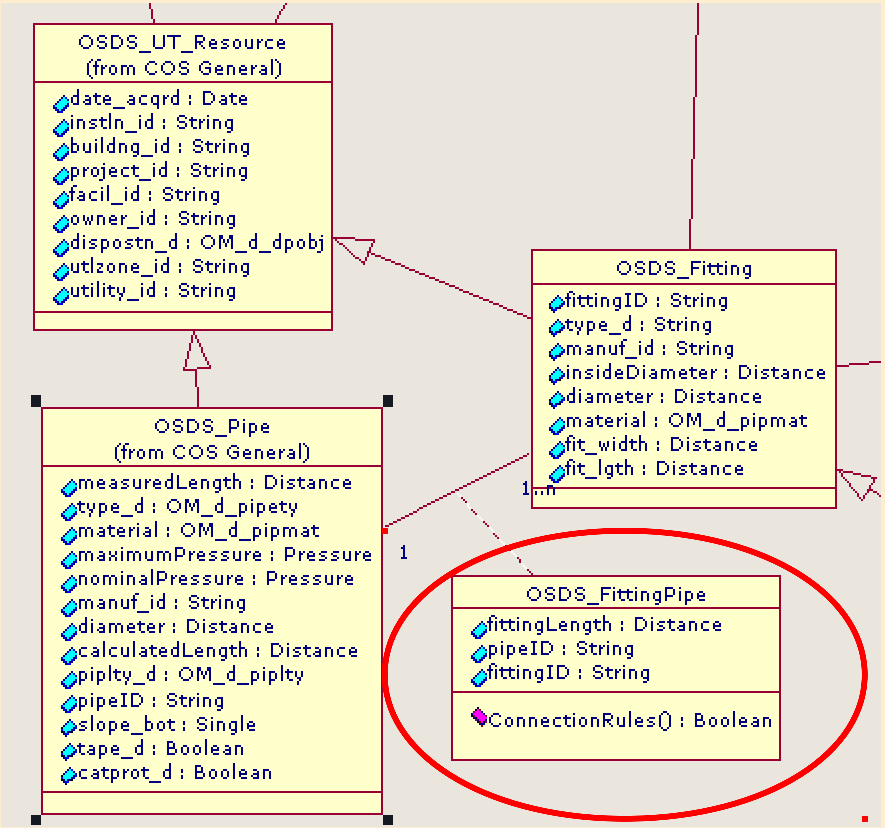
- an ORDER contains at least 1 or more PRODUCTS
- a PRODUCT may be related to 0 or more ORDERS

Association Class

An **Association Class** is a class that is created at the time of the Linking or associating of two classes.

(example : when a pipe is connected to a fitting, a new class is created that can contain information about the connection that does not exist separately for either class).

The IFC classes use this very frequently.



OSDS_UT_Resource
(from COS General)

- date_acqrd : Date
- instln_id : String
- buildng_id : String
- project_id : String
- facil_id : String
- owner_id : String
- dispostn_d : OM_d_dpobj
- utlzone_id : String
- utility_id : String

OSDS_Fitting

- fittingID : String
- type_d : String
- manuf_id : String
- insideDiameter : Distance
- diameter : Distance
- material : OM_d_pipmat
- fit_width : Distance
- fit_lgth : Distance

OSDS_Pipe
(from COS General)

- measuredLength : Distance
- type_d : OM_d_pipety
- material : OM_d_pipmat
- maximumPressure : Pressure
- nominalPressure : Pressure
- manuf_id : String
- diameter : Distance
- calculatedLength : Distance
- piplty_d : OM_d_piplty
- pipeID : String
- slope_bot : Single
- tape_d : Boolean
- catprot_d : Boolean

OSDS_FittingPipe

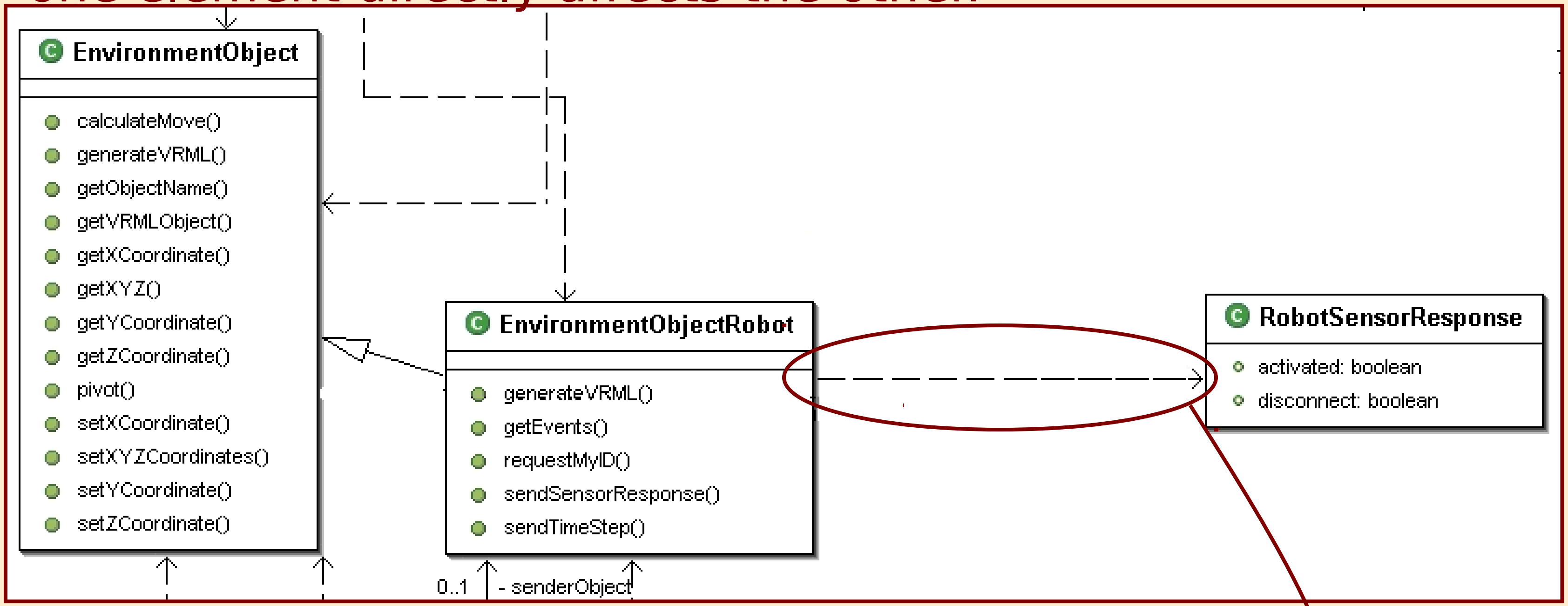
- fittingLength : Distance
- pipeID : String
- fittingID : String

- ConnectionRules() : Boolean

Dependency

Dependency:

relationship between 2 elements where a change in one element directly affects the other.

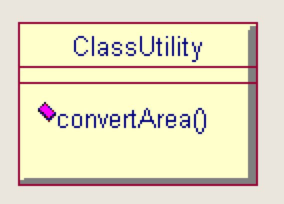


The Robot Sensor Response Object is dependent upon the ROBOT: If the ROBOT is eliminated, the Response Object is eliminated.

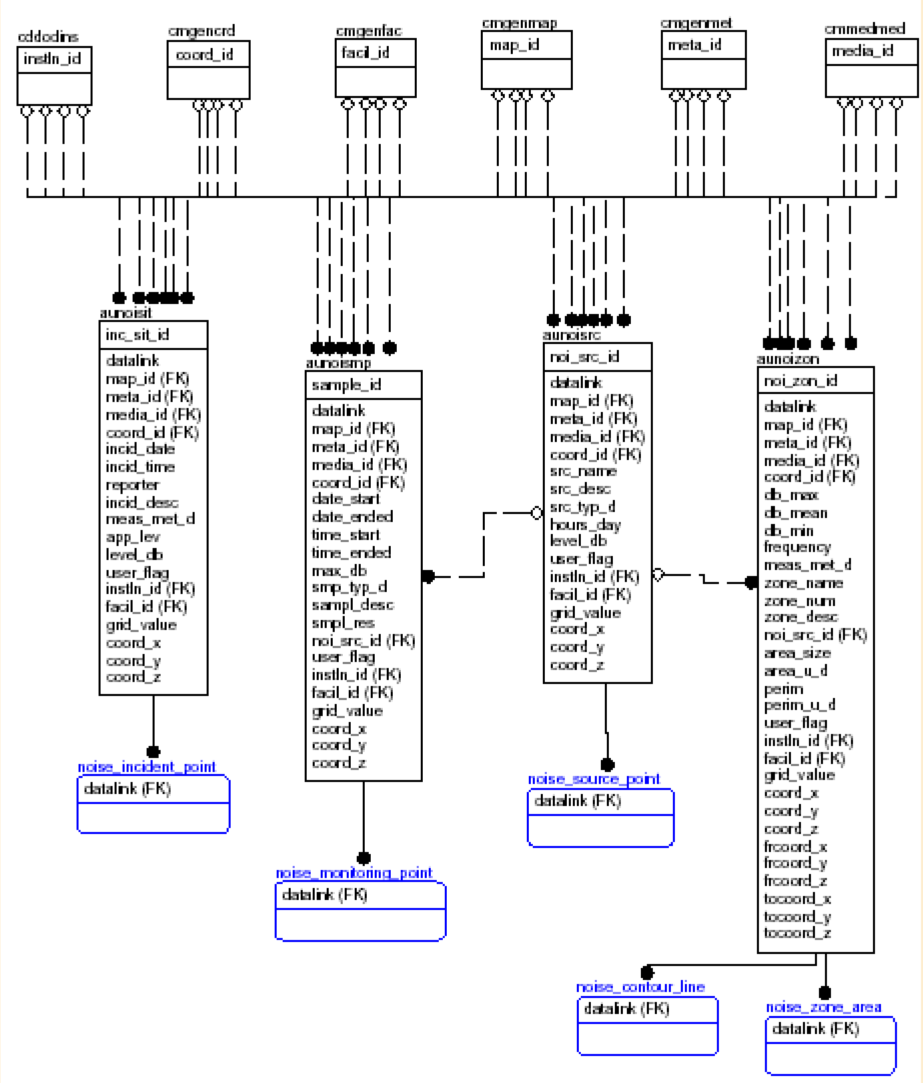
Class Utility

A collection of standard functions used throughout the system:

(conversion routines:acres to square feet, square meters, ...)



IDEF Data Model



cddodins
instln_id

cmgencrd
coord_id

cmgenfac
facil_id

cmgenmap
map_id

cmgenmet
meta_id

cmmedmed
media_id

aunoisit
inc_sit_id
datalink
map_id (FK)
meta_id (FK)
media_id (FK)
coord_id (FK)
incid_date
incid_time
reporter
incid_desc
meas_met_d
app_lev
level_db
user_flag
instln_id (FK)
facil_id (FK)
grid_value
coord_x
coord_y
coord_z

aunoismp
sample_id
datalink
map_id (FK)
meta_id (FK)
media_id (FK)
coord_id (FK)
date_start
date_ended
time_start
time_ended
max_db
smp_typ_d
sampl_desc
smpl_res
noi_src_id (FK)
user_flag
instln_id (FK)
facil_id (FK)
grid_value
coord_x
coord_y
coord_z

aunoisrc
noi_src_id
datalink
map_id (FK)
meta_id (FK)
media_id (FK)
coord_id (FK)
src_name
src_desc
src_typ_d
hours_day
level_db
user_flag
instln_id (FK)
facil_id (FK)
grid_value
coord_x
coord_y
coord_z

aunoizon
noi_zon_id
datalink
map_id (FK)
meta_id (FK)
media_id (FK)
coord_id (FK)
db_max
db_mean
db_min
frequency
meas_met_d
zone_name
zone_num
zone_desc
noi_src_id (FK)
area_size
area_u_d
perim
perim_u_d
user_flag
instln_id (FK)
facil_id (FK)
grid_value
coord_x
coord_y
coord_z
frcoord_x
frcoord_y
frcoord_z
tocoord_x
tocoord_y
tocoord_z

noise_incident_point
datalink (FK)

noise_monitoring_point
datalink (FK)

noise_source_point
datalink (FK)

noise_contour_line
datalink (FK)

noise_zone_area
datalink (FK)

Object-Relational Databases SQL 1999

```
create type Employee
    (person-name varchar(30),
     street varchar(15),
     city varchar(15))
create type Company
    (company-name varchar(15),
    (city varchar(15))
create table employee of Employee
create table company of Company
```

```
create type Works
    (person ref(Employee) scope
     employee,comp ref(Company)
     scope company,
     salary int)
create table works of Works
create type Manages
     (person ref(Employee) scope
      employee,(manager ref(Employee)
      scope employee)
create table manages of Manages
```